МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР ГОУ ВПО «ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра программной инженерии

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ НА ТЕМУ: «Построение аналитических моделей алгоритмов и оценка их сложности» ПО КУРСУ: «Теория алгоритмов и формальных языков»

Выполнил: студент группы ПИ-а

Саевский О.В.

Руководитель:

Коломойцева И.А.

Серёженко О.А.

Щедрин С.В.

Донецк – 2020

**РЕФЕРАТ**

Отчет по курсовой работе содержит: 29 страниц, 12 рисунков, 1 таблица, 3 приложения, 2 источников.

Объект исследования – рекурсивные функции, машины Тьюринга, нормальные алгоритмы Маркова.

Цель – сформировать формальное определение алгоритма в виде трех аналитических моделей, написать программную реализацию машины Тьюринга, распознающей язык L = { wcv│w, v принадлежит {0, 1}\* & v!=w & │w│=2k & │v│=2n & k, n≥0}, построить график временной сложности.

Результат – формальное определение алгоритмов на основе рекурсивных функций, машин Тьюринга и нормальных алгоритмов Маркова, программная реализация машины Тьюринга, распознающей язык L = { wcv│w, v принадлежит {0, 1}\* & v!=w & │w│=2k & │v│=2n & k, n≥0}, график временной сложности машины Тьюринга, файловый вариант протокола работы машины Тьюринга.

**ВВЕДЕНИЕ**

В современной программной инженерии алгоритмы, как методы решения задач, занимают ведущее место по сравнению с традиционной математикой. Причем не важно, существует или нет чистое алгоритмическое решение в абстрактных моделях алгоритмов.

Если решение задачи необходимо, широко используется эвристика, а “доказательством” работоспособности алгоритма является успешное его тестирование.

[1] Алгоритм – это набор предписаний, однозначно определяющих содержание и последовательность выполнения операций для преобразования варьируемых исходных данных в искомый результат. Алгоритмам характерны определённость, результативность, дискретность, массовость и выполнимость операций.

Теория алгоритмов изучает вопросы существования алгоритмов для решения некоторой задачи и выбора наилучшего из существующих. В ходе данной работы будут рассмотрены следующие формальные модели алгоритмов: машины Тьюринга; нормальные алгоритмы Маркова; рекурсивные функции.

Цель работы – это изучение формальных моделей алгоритмов. Также одной из основных задач является программная реализация одноленточной и многоленточной машин Тьюринга, распознающей язык L = { wcv│w, v принадлежит {0, 1}\* & v!=w & │w│=2k & │v│=2n & k, n≥0} , и построения графика временной сложности для них. Для решения данной проблемы выбран язык программирования С#, среда программирования Visual Studio 2019

1 ОПИСАНИЕ АНАЛИТИЧЕСКОЙ МОДЕЛИ АЛГОРИТМА В ВИДЕ ЭЛЕМЕНТАРНЫХ МАШИН ТЬЮРИНГА И КОМПОЗИЦИИ МТ

Машина Тьюринга представляет собой алгоритм как некоторое детерминированное устройство (автомат), реализующий действие над словами. Состоит из бесконечной ленты, разделённой на ячейки. На этой ленте могут быть записаны слова в некотором заранее фиксированном алфавите.

По ленте движется управляющее устройство, обозревающее одну из ячеек. В зависимости от состояния машины и содержания обозреваемой ячейки, машина может, в соответствие с командой, изменить (или не изменять) состояние, заменить (или не заменять) содержимое обозреваемой ячейки, сдвинуть (или не сдвигать) на одну ячейку управляющее устройство влево или вправо.

[2] Существует несколько способов описания машины Тьюринга: - система команд; - функциональная таблица; - диаграмма состояний (граф переходов). Опишем тремя способами машину Тьюринга, вычисляющую x > y. В таблице 1.1 представлена функциональная таблица данной машины.

Таблица 1.1 - Функциональная таблица

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | \* | 1 | T | F | = | λ |
| q1 | q1\*R | q11R | - | - | q2=R | q1 λR |
| q2 | q2\*R | q3\*L | - | - | q2=L | q4 λL |
| q3 | q3\*L | q1\*R | - | - | q3=L | q6 λR |
| q4 | q41L | q51R | - | - | q4=L | qzFE |
| q5 | - | q51L | - | - | q5=R | qzTE |
| q6 | q61R | - | - | - | q6=R | qzFE |

На рисунке 1.1 представлена система команд рассматриваемой машины Тьринга.

|  |  |
| --- | --- |
| q1 \* –> q1\*R  q1 1 –> q11R  q1 = –> q2=R  q1 λ->q1 λ R  q2 \* –> q2\*R  q2 1 –> q3\*L  q2 = –> q2=L  q2 λ –> q4 λL  q3 \* –> q3\*L  q3 1 –> q1\*R  q3 = –> q3=L  q3 λ –> q6 λR | q4 \* –> q41L  q4 1 –> q51R  q4 = –> q4=L  q4 λ –> qzFE  q5 1 –> q51L  q5 = –> q5=R  q5 λ –> qzTE  q6 \* –> q61R  q6 = –> q6=R  q6 λ –> qzFE |

Рисунок 1.1 – Система команд

На рисунке 1.2 изображен граф переходов (диаграмма состояний), но котором вершины графа – это состояния машины Тьюринга, а дуги – переходы между ними.

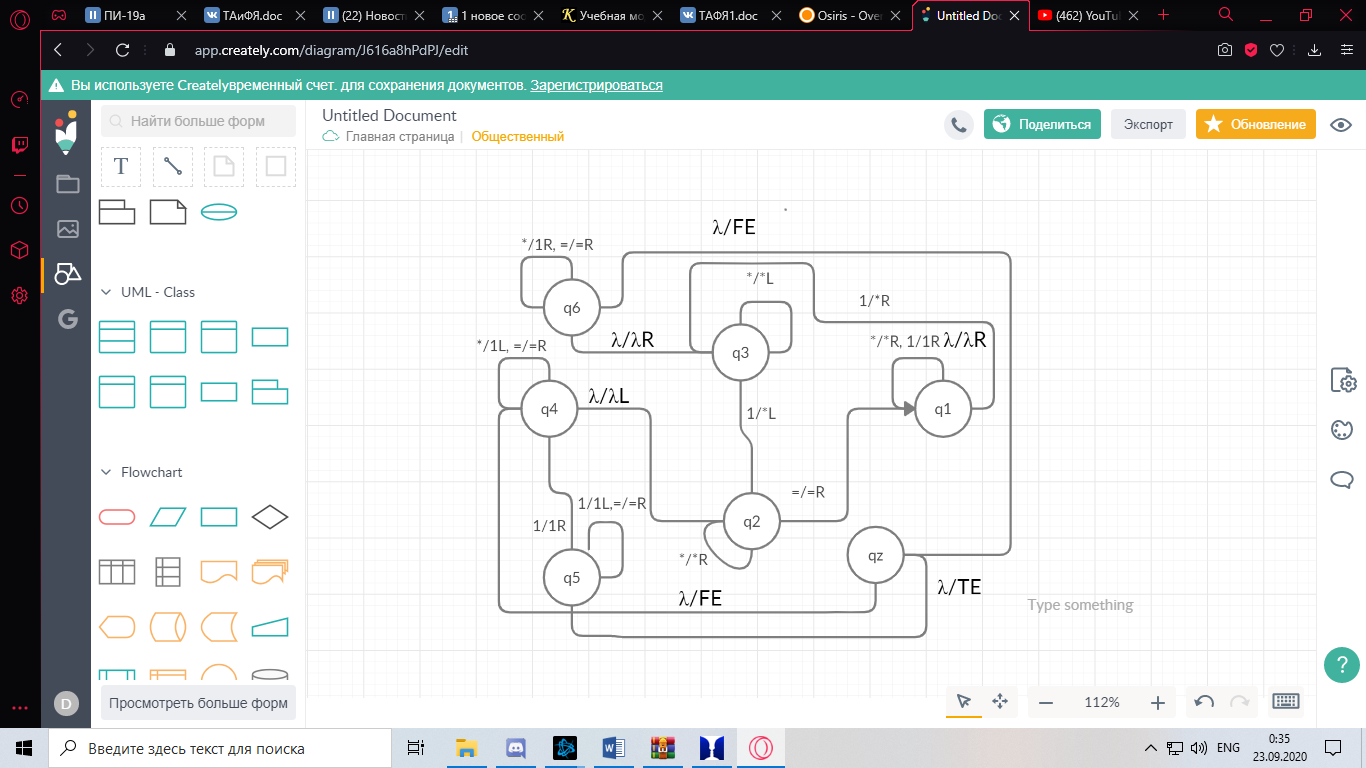


Рисунок 1.2 – Граф переходов

Тестовые примеры работы машины Тьюринга по разным входным данным представлены на рисунке 1.3.

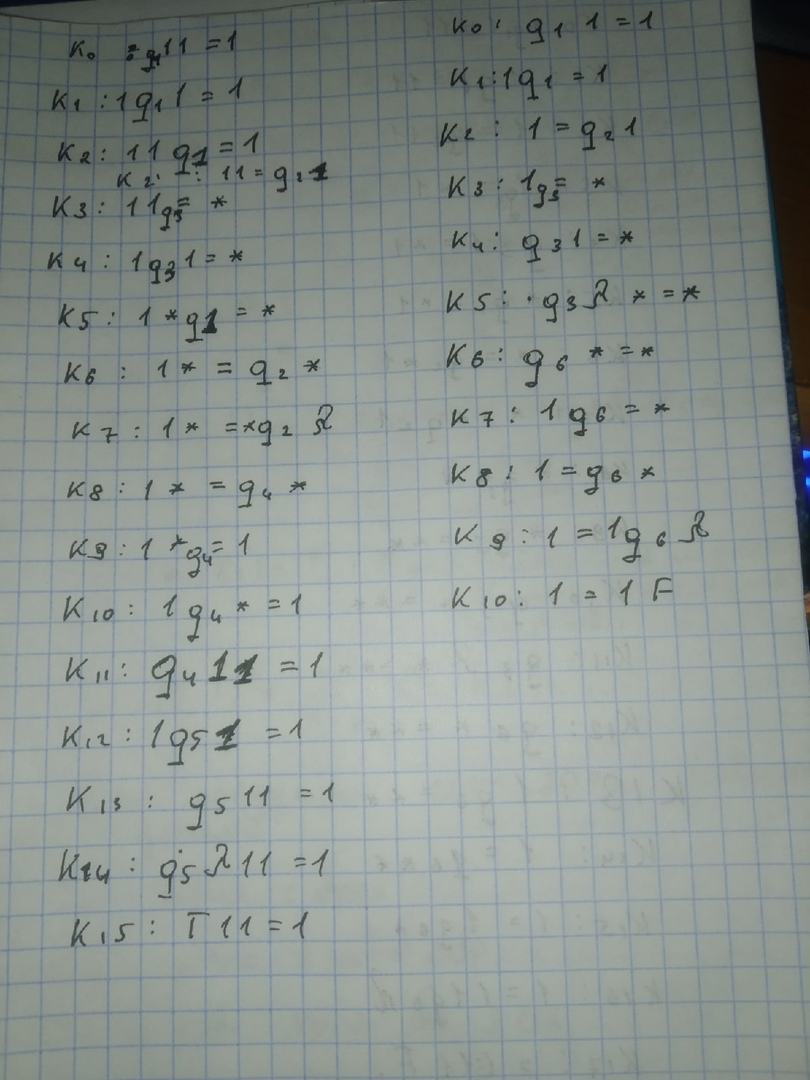


Рисунок 1.3 - Пример работы машины Тьюринга

Составим композицию машин Тьюринга на примере следующей задачи. Найти количество простых делителей числа n. Для более наглядного решения составим блок-схему (см. рис. 1.4)

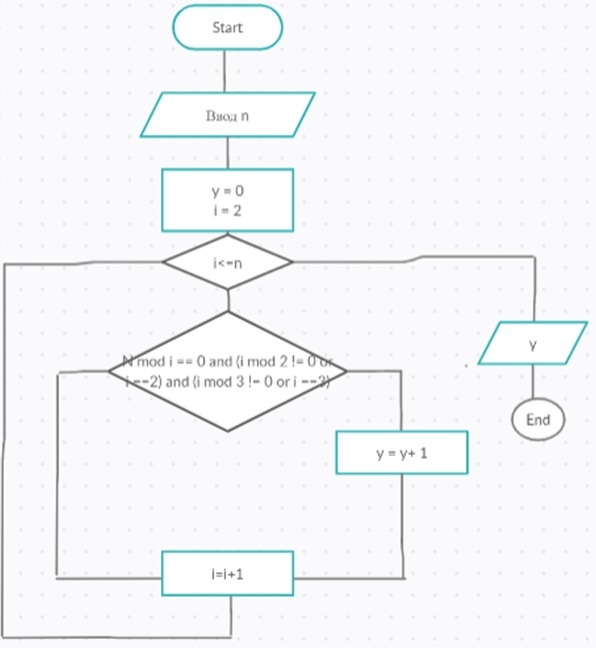


Рисунок 1.4 - Блок-схема алгоритма

M0 – МТ, реализующая копирование входного слова;

M0 – МТ, реализующая функцию установки константы ноль;

Mni – МТ, реализующая функцию выбора i-того аргумента из n аргументов;

M+1 – МТ, реализующая функцию деления аргумента по модулю на n;

M<= - МТ, реализующая предикат i<n.

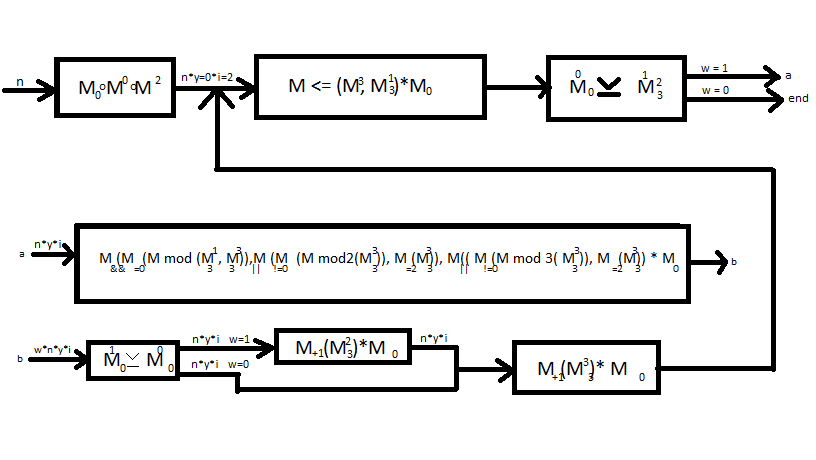


Рисунок 1.5 – Композиция машин Тьюринга

2 РАЗРАБОТКА АНАЛИТИЧЕСКОЙ И ПРОГРАММНОЙ МОДЕЛИ АЛГОРИТМА ДЛЯ РАСПОЗНАЮЩЕЙ МАШИНЫ ТЬЮРИНГА

2.1 Формальное определение распознающей машины Тьюринга Для распознания слов языка L = { wcv│w, v принадлежит {0, 1}\* & v!=w & │w│=2k & │v│=2n & k, n≥0} разработан алгоритм (для одноленточной и многоленточной машин Тьюринга). Рассмотрим его более подробно.

Алгоритм для одноленточной машины Тьюринга (см. рис. 2.1 – 2.2) работает по следующему принципу.

1. q1 0-> q2 0R
2. q1 1->q2 1R
3. q1 c->q14 cL
4. q1 \*->q4 \*L
5. q2 0-> q1 0R
6. q2 1-> q1 1R
7. q2 c->q3 cL
8. q2 \*->q4 \*L
9. q2 λ->qz FE
10. q3 0->q3 0L
11. q3 1-> q3 1L
12. q3 \*->q3 \*L
13. q3 λ->qz FE
14. q4 0->q5 \*R
15. q4 1->q4 \*R
16. q4 c->q6 cR
17. q4 λ->q14 λR
18. q5 c->q8 cR
19. q5 \*->q5 \*R
20. q6 0->q7 0R
21. q6 1->1q7 1R
22. q6 c->q18 cR
23. q6 \*->q10 \*L
24. q6 λ->q10 λ:L
25. q7 0->q6 0R
26. q7 1->q6 1R
27. q7 c->q18 cR
28. q7 \*->q10 \*L
29. q7 λ->qz FE
30. q8 0->q9 0R
31. q8 1->q9 1R
32. q8 c->q18 cR
33. q8 \*->q13 \*L
34. q8 λ->q13 λL
35. q9 0->q8 0R
36. q9 1->q8 1R
37. q9 c->q18 cR
38. q9 \*->q13 \*L
39. q9 λ ->qz FE
40. q10 0->q11 0R
41. q10 1->q12 \*L
42. q10 c->q11 cR
43. q11 0->q11 0R
44. q11 0->q11 1R
45. q11 c->q11 cR
46. q11 \*->q11 \*R
47. q11 λ ->qz TE
48. q12 0->q12 0L
49. q12 1->q12 1L
50. q12 c->q12 cL
51. q12 \*->q12 \*L
52. q12 λ ->q1 λR
53. q13 0->q12 \*L
54. q13 1->q11 1R
55. q13 c->q11 cR
56. q14 0->q4 0E
57. q14 1->q4 1E
58. q14 c-> q15 cR
59. q14 \*->q14 \*R
60. q14 λ ->q14 λ E
61. q15 0->q17 0R
62. q15 1->q17 1R
63. q15 c->q18 cR
64. q15 \*->q15 \*R
65. q15 λ->qz FE
66. q16 0->q17 0R
67. q16 1->q17 1R
68. q16 c->q18 cR
69. q16 \*->q17 \*R
70. q16 λ->qz TE
71. q17 0->q16 0R
72. q17 1->q16 1R
73. q17 c->q18 cR
74. q17 \*->q16 \*R
75. q17 λ->qz FE
76. q18 0->q18 0R
77. q18 1->q18 1R
78. q18 \*->q18 \*R
79. q18 c->q18 cR
80. q18 λ->qz FE

Рисунок 2.1 – Система команд для одноленточной машины Тьюринга

На ленту подаётся слово, после чего каретка начинает своё движение вправо. Каждый символ переходит в другое состояния вплоть до символа “c”, проверяя элемент на четность. Если после “c” встречаем еще одно вхождение символа “c”, слово не входит в алгоритм. Проверяем тем же способом второе слово. Если слова по размеру входят в алфавит, поиск совпадений осуществляется путем запоминания последнего символа в слова и сравнения его с символом другого слова. Если символы отличны – алфавит входит в L, иначе, сравниваем предыдущие символы вплоть до первых.

2.2 Протоколы работы машины Тьюринга

Рассмотрим тестовые примеры работы описанных машин Тьюринга. Для одноленточной машины подадим на вход слова 10c01 (слово не принадлежит языку) (см. рис. 2.5) и 10c1 (слово принадлежит языку) (см. рис. 2.6).

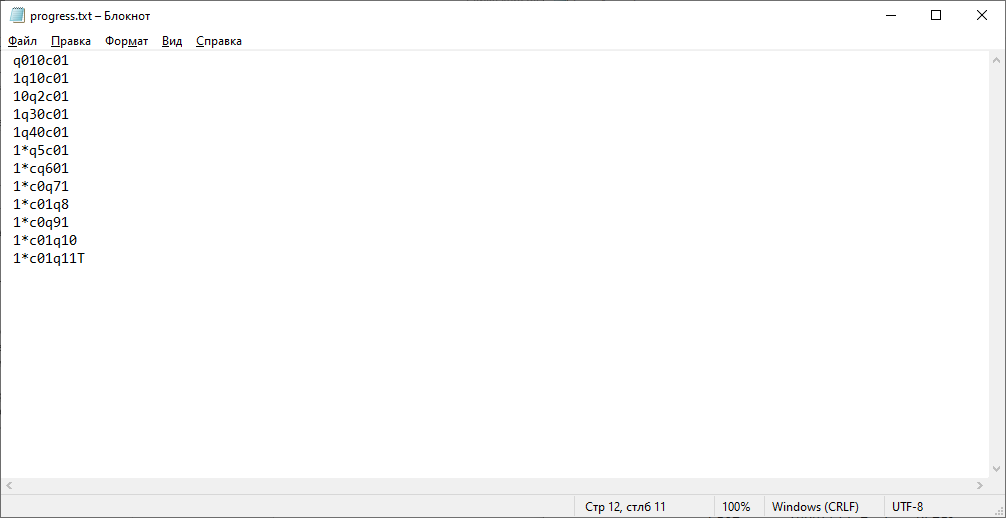


Рисунок 2.2 – Протокол работы машины Тьюринга со словом 10c01

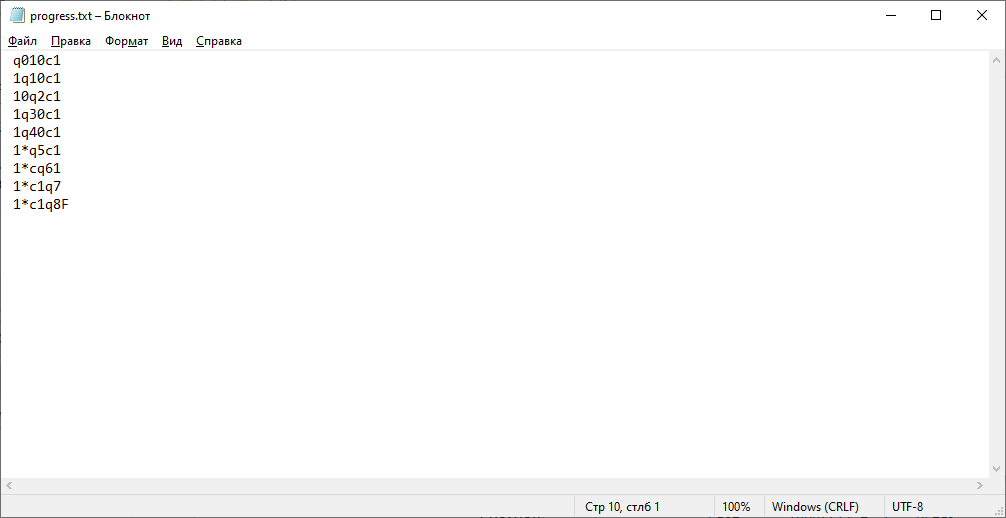


Рисунок 2.3 – Протокол работы машины Тьюринга со словом 10c1

2.3 Программная модель машины Тьюринга

Разработанная программа демонстрирует работу одноленточной и двухленточной машин Тьюринга. Разработана на языке C# в среде программирования Visual Studio 2019. Пользователь вводит слово в текстовое поле, которое может содержать только символы алфавита (реализована проверка на корректность ввода данных). После чего слово конвертируется из String в StringBuilder. Преимущество последнего в том, что в отличие от String объекты StringBuilder можно динамически менять (изменять символы, длину и т.д.).

Все состояния описаны с помощью конструкций switch case. Состояния вызывают друг друга рекурсивно, пока алгоритм не завершится. Для реализации движения каретки влево и вправо использовуется индекс рассматриваемого символа. В зависимости от того, в какую сторону нужно двигаться каретке, к индексу применяется инкремент или декремент. Каждый шаг машины Тьюринга выводится в текстовое поле в виде тестов. Также протокол работы машины записывается в текстовый файл.

2.4 Протоколы работы машины Тьюринга, построенные программно

Рассмотрим тестовые примеры работы реализованных машин Тьюринга. Для одноленточной машины подадим на вход слова 11c011 (слово не принадлежит языку) (см. рис. 2.4) и 11c10 (слово принадлежит языку) (см. рис. 2.5).

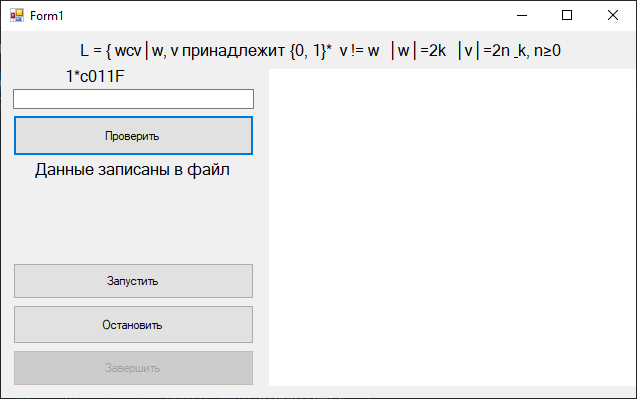


Рисунок 2.4 – Работа машины Тьюринга со словом 11c011

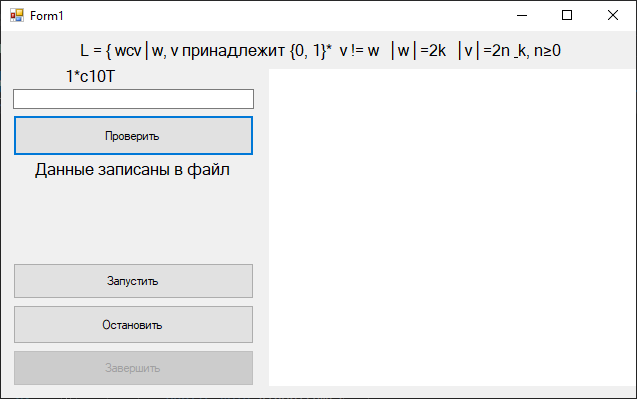


Рисунок 2.5 – Работа машины Тьюринга со словом 11c10

2.5 Расчет временной сложности (график функции временной сложности)

Временная сложность алгоритма – это зависимость времени выполнения алгоритма от объёма входных данных. В случае машины Тьюринга от количества символов во входном слове зависит количество тактов, за которое алгоритм завершит работу.

Функции для нахождения точек и отрисовки графика реализованы в фоновом потоке throw. Для этого была использована компонента chart и её события. В функции PrintGraph реализована функция по расчёту координат для графика.

Координата x – количество символов в рассматриваемом слове, y – максимальное количество тактов. Рассмотрим генерацию слов для одноленточной машины Тьюринга (см. рис. 2)

File.Delete("graph.txt");

Invoke((MethodInvoker)(() =>

{

chart1.Series[0].Points.AddXY(0, 0);

}));

int maxcount = 0;

string maxvalue = null;

const int n = 3;

int m = 1;// n = максимальная цифра

while (true)

{

string col = null;

List<string> all = new List<string>();

char[] temp;

temp = new char[m + 1];

for (int i = 0; i < m + 1; i++)

{

temp[i] = '0';

}

while (temp[0] == '0')

{

char[] tmp = new char[m + 1];

for (int i = 1; i <= m; i++)

{

tmp[i] = temp[i];

}

for (int i = 1; i <= m; i++)

{

col += temp[i].ToString();

}

col = col.Replace('2', 'c');

Turing graph = new Turing(col, false);

Invoke((MethodInvoker)(() =>

{

graph.Algorithm();

}));

if (maxcount < graph.GetCount())

{

maxcount = graph.GetCount();

maxvalue = col;

}

col = null;

temp[m]++;

for (int i = m; i >= 0; i--)

{

if (Convert.ToInt32(temp[i].ToString()) >= n)

{

temp[i] = '0';

temp[i - 1]++;

}

}

}

suspend.WaitOne();

Invoke((MethodInvoker)(() =>

{

StreamWriter fw = new StreamWriter("graph.txt", true);

chart1.Series[0].Points.AddXY(m, maxcount);

chart1.Update();

fw.WriteLine(m.ToString() + " " + maxcount.ToString());

fw.Close();

}));

maxcount = 0;

m++;

}

Рисунок 2.6 - Генерация слов для одноленточной машины Тьюринга

Так как алфавит состоит из 1 и 0 и с, разработан алгоритм по перебору чисел в троичной системе счисления. Переменная m отвечает за количество символов в слове. В конце генерации слова все цифры “2” заменяются символом “c” и запускается алгоритм.

Функция работает, пока пользователь не нажмёт кнопку «Остановить график», тем самым останавив работу фонового потока.

Таким образом на рисунке 2.7 показан график для одноленточной машины Тьюринга.

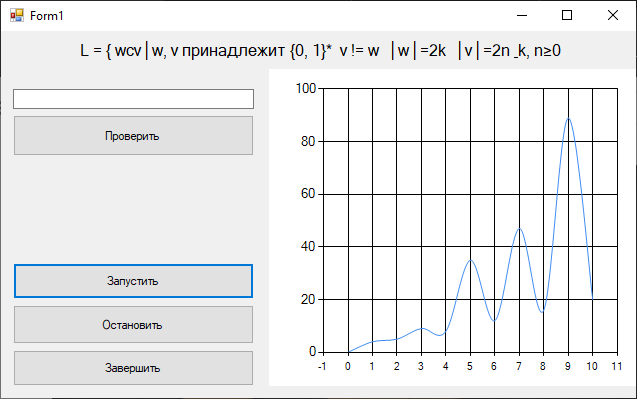


Рисунок 2.7 – График для одноленточной МТ

3 РАЗРАБОТКА АНАЛИТИЧЕСКОЙ МОДЕЛИ АЛГОРИТМА С ИСПОЛЬЗОВАНИЕМ НОРМАЛЬНЫХ АЛГОРИТМОВ МАРКОВА

Нормальные алгоритмы Маркова – это упорядоченный набор элементарных операций над словами. Марковская подстановка – это операция над словами α, β, γ, заключающаяся в следующем. В исходном слове γ ищется самое левое вхождение слова α, если оно существует, α заменяется на β в слове γ. Полученное слово γ' является результатом применения марковской подстановки к слову γ.

Рассмотрим особенности алгоритма на примере. Вычислим x > y в унарном коде с помощью марковской подстановки. Реализация алгоритма показана на рисунке 3.1.

1=1 -> =

11 -> 1

1= -> .T

=1 -> .F

= -> .F

Примеры:

1111=1 -> 111= -> 11= -> 1= -> T

1 = 111 -> =11 -> =1 -> F

11=11 -> 1=1 -> = -> F

4 ОПИСАНИЕ ФОРМАЛЬНОЙ МОДЕЛИ АЛГОРИТМА НА ОСНОВЕ РЕКУРСИВНЫХ ФУНКЦИЙ

Рекурсивный алгоритм – это алгоритм, который в качестве выполняемых операций может использовать обращение к самому себе. При построении рекурсивных функций принят традиционный в теории алгоритмов конструктивный подход: задаётся «базис», то есть несколько простейших, очевидным образом вычислимых функций, и способ построения из них остальных функций с помощью специальных операторов. Опишем на конкретном примере формальную модель алгоритма на основе рекурсивных функций. Найти количество простых делителей числа n.

Рассмотрим тестовый пример работы рекурсивной функции при n = 6

i:2

i:3

i:4

i:5

i:6

**ВЫВОДЫ**

За время выполнения курсовой работы были закреплены навыки работы с такими формальными моделями алгоритмов, как машина Тьюринга, нормальные алгоритмы Маркова и рекурсивные функции. Рассмотрен принцип работы каждой модели на конкретных примерах. Более подробно были изучены особенности работы машины Тьюринга, а также оценка временной сложности алгоритмов. В результате чего разработан программный продукт, имитирующий работу одноленточной машины Тьюринга, для языка L = { wcv│w, v принадлежит {0, 1}\* & v!=w & │w│=2k & │v│=2n & k, n≥0}.

Результат работы программы – вывод ‘F’ или ‘T’ и всплывающее сообщение в зависимости от того принадлежит слово языку или нет соответственно. Также в программе реализованы: запрет ввода с клавиатуры символов не из входного алфавита заданного языка, вывод на экран каждый шаг работы машины Тьюринга и сохранение протокола работы в текстовом файле.

Построение графика временной сложности происходит в отдельном потоке, при этом точки заносятся на график постепенно, по мере расчёта координат. Для их расчёта генерируется слово длиной m методом полного перебора, для каждого из них запускается машина Тьюринга, после чего получаем максимальное число тактов. Осуществляется построение до остановки потока пользователем.

ПЕРЕЧЕНЬ ССЫЛОК

1. Поляков В.И., Скорубский В.И. Основы теории алгоритмов. – СПб: СПб НИУ ИТМО, 2012. – 51 с.

2. Косовская Т.М. Машины Тьюринга [Электронный ресурс]. – Режим доступа: http://ipo.spb.ru/journal/content/626/МАШИНЫ%20ТЬЮРИНГА.pdf

ПРИЛОЖЕНИЕ А

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Приложение Б

Руководство пользователя

Программа предназначена для демонстрации работы машины Тьюринга на примере языка L = { wcv│w, v принадлежит {0, 1}\* & v!=w & │w│=2k & │v│=2n & k, n≥0}. Пользователь может ввести слово (только символы алфавита), проверить принадлежит ли оно языку, при этом данные выводятся в текстовое поле и сохраняются в файл, а также построить график временной сложности. Пользователь может ввести слово, и запустить её МТ, нажав кнопку «Запустить».

О результатах работы алгоритма можно узнать из файла, в котором указано принадлежит ли введенное слово языку или нет. Чтобы построить график, пользователь должен нажать кнопку «Запустить». Для завершения – нажать кнопку «Завершить. Построение графиков не зависит от работы машин Тьюринга. Поэтому работа машины и построение графика может осуществляться параллельно.

Приложение

В Исходный код

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.IO;

using System.Threading;

namespace TaiFYA

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

bool c = true;

Thread thread;

ManualResetEvent suspend = new ManualResetEvent(false);

public class Turing

{

private StreamWriter file;

private static int i;

public char[] input;

public string sinput;

private int count = 0;

private bool write;

public Turing(string text, bool write)

{

input = new char[text.Length + 2];

input[0] = ' ';

for (int v = 0; v < text.Length; v++)

{

input[v + 1] = text[v];

}

input[input.Length - 1] = ' ';

sinput = new string(input);

this.write = write;

if(write == true)

{

file = new StreamWriter("progress.txt");

}

}

private void Write()

{

if (write == true)

{

string sinp = new string(input);

string fileline = (sinp.Insert(i, "q" + count));

file.WriteLine(fileline);

}

}

public void Run()

{

Algorithm();

file.Close();

}

public void Algorithm()

{

i = 1;

q1();

}

private void q1()

{

Write();

count++;

switch (input[i])

{

case '0': i++; q2(); break;

case '1': i++; q2(); break;

case 'c': i--; q14(); break;

case '\*': i--; q4(); break;

case ' ': input[i] = 'F'; Write();

break;

}

}

private void q2()

{

Write();

count++;

switch (input[i])

{

case '0': i++; q1(); break;

case '1': i++; q1(); break;

case 'c': i--; q3(); break;

case '\*': i--; q4(); break;

case ' ': input[i] = 'F'; Write();

break;

}

}

private void q3()

{

Write();

count++;

switch (input[i])

{

case '0': i--; q3(); break;

case '1': i--; q3(); break;

case '\*': i--; q3(); break;

case ' ': input[i] = 'F'; Write();

break;

}

}

private void q4()

{

Write();

count++;

switch (input[i])

{

case '0': input[i] = '\*'; i++; q5(); break;

case '1': input[i] = '\*'; i++; q4(); break;

case 'c': i++; q6(); break;

case '\*': i++; q4(); break;

case ' ': i++; q14(); break;

}

}

private void q5()

{

Write();

count++;

switch (input[i])

{

case 'c': i++; q8(); break;

case '\*': i++; q5(); break;

}

}

private void q6()

{

Write();

count++;

switch (input[i])

{

case '0': i++; q7(); break;

case '1': i++; q7(); break;

case 'c': i++; q18(); break;

case '\*': i--; q10(); break;

case ' ': i--; q10(); break;

}

}

private void q7()

{

Write();

count++;

switch (input[i])

{

case '0': i++; q6(); break;

case '1': i++; q6(); break;

case 'c': i++; q18(); break;

case '\*': i--; q10(); break;

case ' ': input[i] = 'F'; Write();

break;

}

}

private void q8()

{

Write();

count++;

switch (input[i])

{

case '0': i++; q9(); break;

case '1': i++; q9(); break;

case 'c': i++; q18(); break;

case '\*': i--; q13(); break;

case ' ': i--; q13(); break;

}

}

private void q9()

{

Write();

count++;

switch (input[i])

{

case '0': i++; q8(); break;

case '1': i++; q8(); break;

case 'c': i++; q18(); break;

case '\*': i--; q13(); break;

case ' ': input[i] = 'F'; Write();

break;

}

}

private void q10()

{

Write();

count++;

switch (input[i])

{

case '0': i++; q11(); break;

case '1': input[i] = '\*'; i--; q12(); break;

case 'c': i++; q11(); break;

}

}

private void q11()

{

Write();

count++;

switch (input[i])

{

case '0': i++; q11(); break;

case '1': i++; q11(); break;

case 'c': i++; q11(); break;

case '\*': i++; q11(); break;

case ' ': input[i] = 'T'; Write();

break;

}

}

private void q12()

{

Write();

count++;

switch (input[i])

{

case '0': i--; q12(); break;

case '1': i--; q12(); break;

case 'c': i--; q12(); break;

case '\*': i--; q12(); break;

case ' ': i++; q1(); break;

}

}

private void q13()

{

Write();

count++;

switch (input[i])

{

case '0': input[i] = '\*'; i--; q12(); break;

case '1': i++; q11(); break;

case 'c': i++; q11(); break;

}

}

private void q14()

{

Write();

count++;

switch (input[i])

{

case '0': q4(); break;

case '1': q4(); break;

case 'c': i++; q15(); break;

case '\*': i++; q14(); break;

case ' ': i++; q14(); break;

}

}

private void q15()

{

Write();

count++;

switch (input[i])

{

case '0': i++; q17(); break;

case '1': i++; q17(); break;

case 'c': i++; q18(); break;

case '\*': i++; q15(); break;

case ' ': input[i] = 'F'; Write();

break;

}

}

private void q16()

{

Write();

count++;

switch (input[i])

{

case '0': i++; q17(); break;

case '1': i++; q17(); break;

case 'c': i++; q18(); break;

case '\*': i++; q17();break;

case ' ': input[i] = 'T'; Write();

break;

}

}

private void q17()

{

if (write == true)

{

string sinp = new string(input);

string fileline = (sinp.Insert(i, "q" + count));

file.WriteLine(fileline);

}

count++;

switch (input[i])

{

case '0': i++; q16(); break;

case '1': i++; q16(); break;

case 'c': i++; q18(); break;

case '\*': i++; q16(); break;

case ' ': input[i] = 'F'; break;

}

}

private void q18()

{

if (write == true)

{

string sinp = new string(input);

string fileline = (sinp.Insert(i, "q" + count));

file.WriteLine(fileline);

}

count++;

switch (input[i])

{

case '0': i++; q18(); break;

case '1': i++; q18(); break;

case '\*': i++; q18(); break;

case 'c': i++; q18(); break;

case ' ': input[i] = 'F'; break;

}

}

public int GetCount()

{

return count;

}

}

BackgroundWorker bw;

public void PrintGraph()

{

File.Delete("graph.txt");

Invoke((MethodInvoker)(() =>

{

chart1.Series[0].Points.AddXY(0, 0);

}));

int maxcount = 0;

string maxvalue = null;

const int n = 3;

int m = 1;// n = максимальная цифра

while (true)

{

string col = null;

List<string> all = new List<string>();

char[] temp;

temp = new char[m + 1];

for (int i = 0; i < m + 1; i++)

{

temp[i] = '0';

}

while (temp[0] == '0')

{

char[] tmp = new char[m + 1];

for (int i = 1; i <= m; i++)

{

tmp[i] = temp[i];

}

for (int i = 1; i <= m; i++)

{

col += temp[i].ToString();

}

col = col.Replace('2', 'c');

Turing graph = new Turing(col, false);

Invoke((MethodInvoker)(() =>

{

graph.Algorithm();

}));

if (maxcount < graph.GetCount())

{

maxcount = graph.GetCount();

maxvalue = col;

}

col = null;

temp[m]++;

for (int i = m; i >= 0; i--)

{

if (Convert.ToInt32(temp[i].ToString()) >= n)

{

temp[i] = '0';

temp[i - 1]++;

}

}

}

suspend.WaitOne();

Invoke((MethodInvoker)(() =>

{

StreamWriter fw = new StreamWriter("graph.txt", true);

chart1.Series[0].Points.AddXY(m, maxcount);

chart1.Update();

fw.WriteLine(m.ToString() + " " + maxcount.ToString());

fw.Close();

}));

maxcount = 0;

m++;

}

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

}

private void Form1\_Load(object sender, EventArgs e)

{

chart1.Series[0].ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Spline;

button2.Enabled = false;

}

private void textBox1\_KeyDown(object sender, KeyEventArgs e)

{

}

private void textBox1\_KeyPress(object sender, KeyPressEventArgs e)

{

e.Handled = true;

if (e.KeyChar == '1'|| e.KeyChar == '0' || e.KeyChar == 'c' || e.KeyChar == '\b')

e.Handled = false;

}

private void button1\_Click(object sender, EventArgs e)

{

string result = null;

Turing machine = new Turing(textBox1.Text, true);

machine.Run();

for(int i = 0; i < machine.input.Length; i++)

{

result += machine.input[i];

}

textBox1.Text = null;

label3.Text = result;

label2.Text = "Данные записаны в файл";

}

private void chart1\_Click(object sender, EventArgs e)

{

}

private void button3\_Click(object sender, EventArgs e)

{

/\*Thread thread = new Thread(() =>

{

Invoke((Action)(() =>

{

PrintGraph();

}));

});\*/

button2.Enabled = true;

if (thread == null)

{

thread = new Thread(PrintGraph);

thread.Start();

}

suspend.Set();

}

private void button2\_Click(object sender, EventArgs e)

{

thread.Abort();

thread = null;

chart1.Series[0].Points.Clear();

button2.Enabled = false;

}

private void button4\_Click(object sender, EventArgs e)

{

suspend.Reset();

}

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

if(thread != null)

thread.Abort();

}

private void button5\_Click(object sender, EventArgs e)

{

}

}

}